Adobe® Marketing Cloud

BlackBerry 10 SDK 4.x for Marketing Cloud Solutions

Contents

BlackBerry 10 SDK 4.x for Marketing Cloud Solutions.....3

Developer Quick Start.....4

Adobe Mobile Class and Method Reference.....6

Analytics.....11

Video Analytics.....14

Lifecycle Metrics.....18

Using Bloodhound to Test Mobile Applications.....21

Contact and Legal Information.....22

BlackBerry 10 SDK 4.x for Marketing Cloud Solutions

BlackBerry 10 SDK 4.x for Marketing Cloud Solutions lets you measure native BlackBerry 10 applications using Adobe Analytics.

Getting Started

Complete the tasks in *Developer Quick Start* to add the SDK to your project. Next, review the *Adobe Mobile Class and Method Reference* and then implement *Analytics*.

Supported Versions

BlackBerry 10 or later.

Release History

See the *BlackBerry* release notes.

New Features in Version 4

In addition to significant performance increases, version 4 adds the following new features:

Feature	Description
Opt-in/Opt-out	Quickly enable and disable analytics.

Documentation Revision History

Revision Date	Description
November 1, 2013	Initial release.

Developer Quick Start

This guide walks you through the steps to implement the BlackBerry library.

• *Get the SDK*
• *Add the SDK to your Project*
• *Update The ADBMobileConfig.json Config File*

Get the SDK
The SDK requires BlackBerry 10 or later

After unzipping the downloaded SDK, you'll have the following files in an `AdobeMobile` folder:

- `Device-Coverage/libADBMobileShared.so`
- `Device-Debug/libADBMobileShared.so`
- `Device-Profile/libADBMobileShared.so`
- `Device-Release/libADBMobileShared.so`
- `public/ADBMediaAnalytics.hpp`
- `public/ADBMediaSharedHeader.hpp`
- `public/ADBMediaState.hpp`
- `public/ADBMobile.hpp`
- `Simulator-Coverage/libADBMobileShared.so`
- `Simulator-Debug/libADBMobileShared.so`
- `Simulator-Profile/libADBMobileShared.so`

Add the SDK to your Project

1. Right-click on your project and select "**Configure** > **Add Library**.
2. Select **External library** and click **Next**.
3. Click **Browse** next to the**Device library** field.
4. Navigate to the `ADBMobile-4.0.0BETA-BlackBerry` folder.
5. In the `Device-Debug` folder, select `libADBMobileShared.so` and click **Open**.
6. Click **Browse** next to the**Simulator library** field.
7. Navigate to the `ADBMobile-4.0.0BETA-BlackBerry` folder.
8. In the `Device-Debug` folder, select `libADBMobileShared.so` and click **Open**.
9. Click **Add** next to the**Include folders** field.
10. Navigate to the `ADBMobile-4.0.0BETA-BlackBerry` folder.
11. Add the `public` folder to your includes.
12. In the `ADBMobile-4.0.0BETA-BlackBerry` folder, there is a `.json` config file named `ADBMobileConfig.json`. Copy that file into the root of your project.
13. Right-click on your project and select **Refresh**. The `.json` file should now be visible in your **Project Explorer**.
14. Open the `bar-descriptor.xml` file for your project.
15. At the bottom of the window select the **Assets** tab.
16. Confirm that **(All Configurations)** is selected, then click the **Add Files** in the **Assets** section of the window. (Note, there is a bug in the QNX Momentics IDE that sometimes prevents those buttons from being visible. If you can't see the buttons, resize the windows until they appear).
17. Click the **Workspace** button.
18. Find the `ADBMobileConfig.json` file in your project and click **OK**.

Your application can import the classes/interfaces from the `adobeMobileLibrary.jar` library by using `#include <ADBMobile.hpp>`.

Add App Permissions

In `bar-descriptor.xml` in the project directory, add the line `<permission>access_internet</permission>`, or in the QNX Momentics IDE, check the box for **Internet** in the permissions section of the **Application** tab.

Update The ADBMobileConfig.json Config File

The `ADBMobileConfig.json` file contains global SDK settings. You need to update a few values to get started.

The following is an example of an `ADBMobileConfig.json` file:

```
{
    "version" : "1.0",
    "analytics" : {
        "rsids" : "coolApp",
        "server" : "my.CoolApp.com",
        "charset" : "UTF-8",
        "ssl" : false,
        "offlineEnabled" : true,
        "lifecycleTimeout" : 5,
        "privacyDefault" : "optedin",
    }
}
```

At a minimum, update the `rsids` and `server` parameters.

For more details, see *ADBMobileConfig.json Config File Reference*.

That's it! You're now ready to implement Analytics in your BlackBerry 10 app.

Where to go from here:

- *Adobe Mobile Class and Method Reference*
- *Analytics*

Adobe Mobile Class and Method Reference

Classes and methods provided by the BlackBerry library.

The SDK currently has support for Adobe Analytics. Methods are contained in separate classes according to the solution. The list of solutions and each class is as follows:

Solution	Class
Analytics	Analytics

SDK Settings

Method	Description

getPrivacyStatus	Returns the enum representation of the privacy status for current user.
	• `ADBMobilePrivacyStatusOptIn` - hits are sent immediately.
	• `ADBMobilePrivacyStatusOptOut` - hits are discarded.
	• `ADBMobilePrivacyStatusUnknown` - If your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (then hits are sent) or opt-out (then hits are discarded). If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt in.
	Default: The default value is set in *`ADBMobileConfig.json`*
	Syntax:
	`static ADBMobilePrivacyStatus getPrivacyStatus();`
	Example:
	`ADBMobilePrivacyStatus privacyStatus = ADBMobile::getPrivacyStatus();`

setPrivacyStatus	Sets the privacy status for the current user to `status`. Set to one of the following values:
	• `ADBMobilePrivacyStatusOptIn` - hits are sent immediately.
	• `ADBMobilePrivacyStatusOptOut` - hits are discarded.
	• `ADBMobilePrivacyStatusUnknown` - If your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (then hits are sent) or opt-out (then hits are discarded). If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt in.
	Syntax:
	`static void setPrivacyStatus(ADBMobilePrivacyStatus status);`
	Example:
	`ADBMobile::setPrivacyStatus(ADBMobilePrivacyStatusOptIn);`

getUserIdentifier	Returns the user identifier if a custom identifier has been set. Returns null if a custom identifier is not set. Default: `null` **Syntax:** `static QString getUserIdentifier();`

Method	Description
	Example: `QString userId = ADBMobile::getUserIdentifier();`

setUserIdentifier	Sets the user identifier to `identifier`.
	Syntax:
	`static void setUserIdentifier(QString identifier);`
	Example:
	`ADBMobile::setUserIdentifier("billybob");`

getDebugLogging	Returns the current debug logging preference.
	Default: false
	Syntax:
	`static bool getDebugLogging();`
	Example:
	`bool debugging = ADBMobile::getDebugLogging();`

setDebugLogging	Sets the debug logging preference to `debugLogging`.
	Syntax:
	`static void setDebugLogging(bool debugLogging);`
	Example:
	`ADBMobile::setDebugLogging(true);`

collectLifecycleData	Indicates to the SDK that lifecycle data should be collected for use across all solutions in the SDK. See *Lifecycle Metrics*.
	Syntax:
	`static void collectLifecycleData();`
	Example:
	`ApplicationUI::ApplicationUI(bb::cascades::Application *app): QObject(app)`
	`{`
	`//...`
	`ADBMobile::collectLifecycleData();`
	`}`

Analytics Methods

Each of these methods is used to send data into your Adobe Analytics report suite.

Method	Description
trackState	Tracks an app state with optional context data. States are the views that are available in your app, such as "home dashboard", "app settings", "cart", and so on. These states are similar to pages on a website, and `trackState` calls increment page views.

Method	Description

	 ***Note:*** *This is the only tracking call that increments page views.*
	Syntax:
	`static void trackState(QString state, QHash<QString, QString> contextData = QHash<QString, QString>());`
	Example:
	`ADBMobile::trackState("loginScreen", null);`

trackAction	Tracks an action in your app. Actions are the things that happen in your app that you want to measure, such as "logons", "banner taps", "feed subscriptions", and other metrics.
	Syntax:
	`static void trackAction(QString action, QHash<QString, QString> contextData = QHash<QString, QString>());`
	Example:
	`ADBMobile::trackAction("heroBannerTouched", null);`

trackLocation	Sends the current x y coordinates. Replace event with the event that is received from the subscriber to BPS.
	Syntax:
	`static void trackLocation(bps_event_t *geoEvent, QHash<QString, QString> contextData = QHash<QString, QString>());`
	Example:
	`ADBMobile::trackLocation(event, null);`

ADBMobileConfig.json Config File Reference

The `ADBMobileConfig.json` must be placed in the `assets` folder.

Variable	Description

rsids	(Required) One or more report suites to receive Analytics data. Multiple report suite IDs should be comma-separated with no space between.
	`"rsids" : "rsid"`
	`"rsids" : "rsid1,rsid2"`

server	(Required). Analytics server. This variable should be populated with the server domain, without an "http://" or https://" protocol prefix. The protocol prefix is handled automatically by the library based on the `ssl` variable. If `ssl` is `true`, a secure connection is made to this server. If `ssl` is `false`, a non-secure connection is made to this server.

Variable	Description
charset	Defines the character set you are using for the data sent to Analytics. The charset is used to convert incoming data into UTF-8 for storage and reporting. See *Using the charSet Property*.
ssl	Default: false Enables (true) or disables (false) sending measurement data via SSL (HTTPS).
offlineEnabled	Default: false When enabled (true), hits are queued while the device is offline and sent later when the device is online. Your report suite must be timestamp-enabled to use offline tracking. ***Important:*** *If timestamps are enabled on your report suite, your `offlineEnabled` configuration property must be true. if your report suite is not timestamp enabled, your `offlineEnabled` configuration property must be false. If this is not configured correctly, data will be lost. If you are not sure if a report suite is timestamp enabled, contact Customer Care.* If you are currently reporting AppMeasurement data to a report suite that also collects data from JavaScript, you might need to set up a separate report suite for mobile data, or include a custom timestamp on all JavaScript hits using the `s.timestamp` variable.
lifecycleTimeout	Default: 300 seconds Specifies the length of time, in seconds, that must elapse between app launches before the launch is considered a new session. This timeout also applies when your application is sent to the background and reactivated. The time that your app spends in the background is not included in the session length.
batchLimit	Default: 0 (No limit) Maximum number of offline hits stored in the queue.
privacyDefault	Default: `optedin` • `optedin` - hits are sent immediately. • `optedout` - hits are discarded. • `optunknown` - If your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (then hits are sent) or opt-out (then hits are discarded). If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt in. This sets the initial value only. If this value is ever set or changed in code, then the new value is used going forward until it is changed, or the app is uninstalled and then reinstalled.

The following is an example of an `ADBMobileConfig.json` file:

```
{
    "version" : "1.0",
    "analytics" : {
        "rsids" : "coolApp",
        "server" : "my.CoolApp.com",
```

```
        "charset" : "UTF-8",
        "ssl" : false,
        "offlineEnabled" : true,
        "lifecycleTimeout" : 5,
        "privacyDefault" : "optedin",
    }
}
```

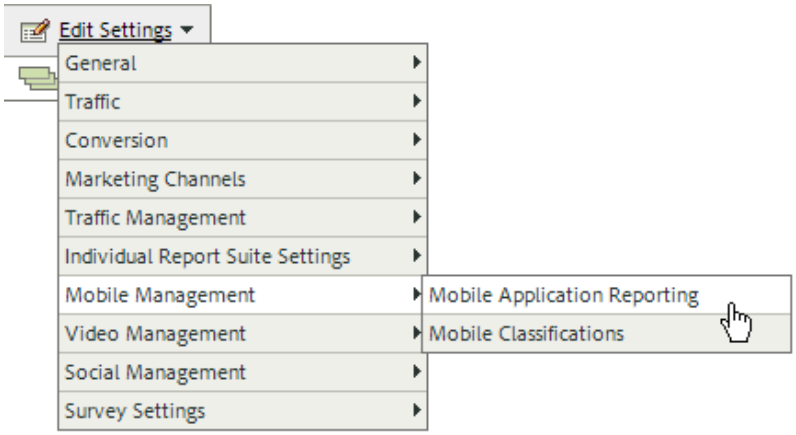
Analytics

After you add the library to your project, you can make any of the Analytics method calls anywhere in your App (make sure you import `ADBMobile.h` to your class).

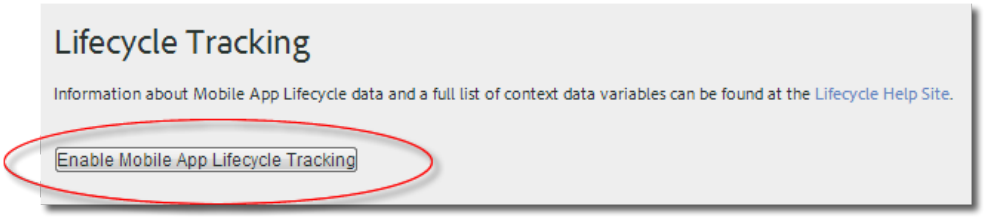
Enable Mobile Application Reports in Analytics

Before you add code, have your Analytics Administrator complete the following to enable Mobile App Lifecycle tracking. This ensures that your report suite is ready to capture metrics as you begin development.

1. Open **Admin Tools** > **Report Suites** and select your mobile report suite(s).
2. Click **Edit Settings** > **Mobile Management** > **Mobile Application Reporting**.



3. Click **Enable Mobile App Livecycle Tracking**, and optionally, **Enable Mobile Location Tracking** and **Enable Google Play Campaign Tracking**.



Lifecycle metrics are now ready to be captured, and **Mobile Application Reports** appear in the **Reports** menu in the marketing reports interface.

Lifecycle Metrics

To collect lifecycle metrics in your app, call `collectLifecycleData()` in the `ApplicationUI` constructor. For example:

```
ApplicationUI::ApplicationUI(bb::cascades::Application *app): QObject(app) {
//...
ADBMobile::collectLifecycleData();
}
```

If `collectLifecycleData()` is called twice in the same session, then your application will report a crash on every call after the first. The SDK sets a flag when the application is shutdown that indicates a successful exit. If this flag is not set, `collectLifecyleData()` reports a crash.

Events, Props, and eVars

If you've looked at the *Adobe Mobile Class and Method Reference*, you are probably wondering where to set events, eVars, props, heirs, and lists. In version 4, you can no longer assign those types of variables directly in your app. Instead, the SDK uses context data and processing rules to map your app data to Analytics variables for reporting.

Processing rules provide you several advantages:

- You can change your data mapping without submitting an update to the App Store.
- You can use meaningful names for data instead of setting variables that are specific to a report suite.
- There is little impact to sending in extra data. These values won't appear in reports until they are mapped using processing rules.

Any values that you were assigning directly to variables should be added to the `data` HashMap instead.

Processing Rules

Processing rules are used to copy the data you send in context data variables to evars, props, and other variables for reporting.

Processing Rules Training @ Summit 2013

Processing Rules Help

Become authorized to use Processing Rules

We recommend grouping your context data variables using "namespaces", as it helps you keep logical ordering. For example, if you want to collect info about a product, you might define the following variables:

```
"product.type":"hat"
"product.team":"mariners"
"product.color":"blue"
```

Context data variables are sorted alphabetically in the processing rules interface, so namespaces let you quickly see variables that are in the same namespace.

Also, we have heard that some of you are naming context data keys using the evar or prop number:

```
"eVar1":"jimbo"
```

This might make it *slightly* easier when you perform the one time mapping in processing rules, but you lose readability during debugging and future code updates can be more difficult. Instead, we strongly recommend using descriptive names for keys and values:

```
"username":"jimbo"
```

Context variables that define counter events can have the same key and value:

```
"logon":"logon"
```

Context data variables that define incrementor events can have the event as the key and the amount to increment as the value:

```
"levels completed":"6"
```

Note: *Adobe reserves the namespace "a.". Aside from that small restriction, context data variables just need to be unique in your login company to avoid collisions.*

(Optional) Enable Offline Tracking

To store hits when the device is offline, you can enable offline tracking in the *ADBMobileConfig.json Config File Reference*. Pay very close attention to the timestamp requirements described in the config file reference before you enable offline tracking.

Analytics Methods

Each of these methods is used to send data into your Adobe Analytics report suite.

Method	Description

trackState	Tracks an app state with optional context data. States are the views that are available in your app, such as "home dashboard", "app settings", "cart", and so on. These states are similar to pages on a website, and `trackState` calls increment page views.
	
	Note: *This is the only tracking call that increments page views.*
	
	Syntax:
	`static void trackState(QString state, QHash<QString, QString> contextData = QHash<QString, QString>());`
	
	Example:
	`ADBMobile::trackState("loginScreen", null);`

trackAction	Tracks an action in your app. Actions are the things that happen in your app that you want to measure, such as "logons", "banner taps", "feed subscriptions", and other metrics.
	
	Syntax:
	`static void trackAction(QString action, QHash<QString, QString> contextData = QHash<QString, QString>());`
	
	Example:
	`ADBMobile::trackAction("heroBannerTouched", null);`

trackLocation	Sends the current x y coordinates. Replace event with the event that is received from the subscriber to BPS.
	
	Syntax:
	`static void trackLocation(bps_event_t *geoEvent, QHash<QString, QString> contextData = QHash<QString, QString>());`
	
	Example:
	`ADBMobile::trackLocation(event, null);`

Video Analytics

Video measurement is described in detail in the *Measuring Video in Analytics* guide. The general process to measure video is very similar across all AppMeasurement platforms. This quick start section provides a basic overview of the developer tasks along with code samples.

Map Player Events to Analytics Variables

The following table lists the media data that is sent to Analytics. Use processing rules to map the context data in the Context Data Variable column to an Analytics variable as described in the Variable Type column.

Context Data Variable	Variable Type	Description
a.media.name	eVar Default expiration: Visit Custom Insight (s.prop, used for video pathing)	(Required) Collects the name of the video, as specified in the implementation, when a visitor views the video in some way. You can add classifications for this variable. (Optional) The Custom Insight variable provides video pathing information.
a.media.name	Custom Insight (s.prop)	(Optional) Provides video pathing information. Pathing must be enabled for this variable by ClientCare. Event type: Custom Insight (s.prop)
a.media.segment	eVar Default expiration: Page view	(Required) Collects video segment data, including the segment name and the order in which the segment occurs in the video. This variable is populated by enabling the `segmentByMilestones` variable when tracking player events automatically, or by setting a custom segment name when tracking player events manually. For example, when a visitor views the first segment in a video, SiteCatalyst might collect the following in the Segments eVar: `1:M:0-25` The default video data collection method collects data at the following points: video start (play), segment begin, and video end (stop). Analytics counts the first segment view at the start of the segment, when the visitor starts watching. Subsequent segment views as the segment begins.
a.contentType	eVar Default expiration: Page view	Collects data about the type of content viewed by a visitor. Hits sent by video measurement are assigned a content type of "video".

Context Data Variable	Variable Type	Description
		This variable does not need to be reserved exclusively for video tracking. Having other content report content type using this same variable lets you analyze the distribution of visitors across the different types of content. For example, you could tag other content types using values such as “article" or “product page" using this variable. From a video measurement perspective, Content Type lets you identify video visitors and thereby calculate video conversion rates.
a.media.timePlayed	Event Type: Counter	Counts the time, in seconds, spent watching a video since the last data collection process (image request).
a.media.view	Event Type: Counter	Indicates that a visitor has viewed some portion of a video. However, it does not provide any information about how much, or what part, of a video the visitor viewed.
a.media.segmentView	Event Type: Counter	Indicates that a visitor has viewed some portion of a video segment. However, it does not provide any information about how much, or what part, of a video the visitor viewed.
a .media.complete	Event Type: Counter	Indicates that a user has viewed a complete video. By default, the complete event is measured 1 second before the end of the video. During implementation, you can specify how many seconds from the end of the video you would like to consider a view complete. For live video and other streams that don't have a defined end, you can specify a custom point to measure completes. For example, after a specific time viewed.

Track Player Events

To measure video playback, The `mediaPlay`, `mediaStop`, and `mediaClose` methods need to be called at the appropriate times. For example, when the player is paused, `mediaStop`. `mediaPlay` is called when playback starts or is resumed.

Media Measurement Class and Method Reference

Method	Description
open	Opens a video for tracking.

Method	Description

	Syntax:
	`void open(QString name, double length, QString playerName, QString playerID = QString());`
	Example:
	`ADBMediaAnalytics::sharedInstance()->open("name", 10.0, "playerName", "playerID");`

openAd	Opens a `MediaSettings` object for tracking.
	Syntax:
	`void openAd(QString name, double length, QString playerName, QString parentName, QString parentPod, double parentPodPosition, QString CPM);`
	Example:
	`ADBMediaAnalytics::sharedInstance()->openAd("name", 10, "playerName", "parentName", "podName", 0, "CPM");`

close	Closes the media item named *name*.
	Syntax:
	`void close(QString name);`
	Example:
	`ADBMediaAnalytics::sharedInstance()->close("name");`

play	Plays the media item named *name* at the given *offset* (in seconds).
	Syntax:
	`void play(QString name, double offset);`
	Example:
	`ADBMediaAnalytics::sharedInstance()->play("name", 0);`

complete	Manually mark the media item as complete at the *offset* provided (in seconds).
	Syntax:
	`void complete(QString name, double offset);`
	Example:
	`ADBMediaAnalytics::sharedInstance()->complete("name", 0);`

stop	Notifies the media module that the video has been stopped or paused at the given *offset*.
	Syntax:
	`stop(QString name, double offset);`
	Example:
	`ADBMediaAnalytics::sharedInstance()->stop("name", 0);`

Method	Description

click	Notifies the media module that the media item has been clicked.
	Syntax:
	`void click(QString name, double offset);`
	Example:
	`ADBMediaAnalytics::sharedInstance()->close("name", 0);`

track	Sends a track action call (no page view) for the current media state.
	Syntax:
	`track(QString name, QHash<QString, QString> additionalData = QHash<QString, QString>());`
	Example:
	`ADBMediaAnalytics::sharedInstance()->track("name", null);`

Lifecycle Metrics

Lists the metrics and dimensions that can be measured automatically by the mobile library.

Lifecycle Metrics and Dimensions

When configured, lifecycle metrics are sent in context data parameters to Analytics. The context data that is sent with each lifecycle tracking call is automatically captured in and reported using the metric or dimension listed in the first column, with the exceptions noted in the description column.

Metrics

Metric	Analytics Context Data Parameter	Description
First Launches	a.InstallEvent	Triggered on first run after installation (or re-installation).
Upgrades	a.UpgradeEvent	Triggered on first run after upgrade (anytime the version number changes).
Daily Engaged Users	a.DailyEngUserEvent	Triggered when the application is used on a particular day. ***Note:*** *Daily Engaged Users is not automatically stored in an Analytics metric. You must create a processing rule that sets a custom event to capture this metric.*
Monthly Engaged Users	a.MonthlyEngUserEvent	Triggered when the application is used during a particular month. ***Note:*** *Monthly Engaged Users is not automatically stored in an Analytics metric. You must create a processing rule that sets a custom event to capture this metric.*
Launches	a.LaunchEvent	Triggered on every run, including crashes and installs. Also triggered on a resume from background when the lifecycle session timeout has been exceeded.
Crashes	a.CrashEvent	Triggered when the application does not exit gracefully. Event is sent on application start after crash (the application is considered to crash if quit is not called).
Previous Session Length	a.PreviousSessionLength	Aggregated total Previous Session Length in seconds.

Dimensions

Dimension	Analytics Context Data Parameter	Description
Install Date	a.InstallDate	Date of first launch after installation. `MM/DD/YYYY`
App ID	a.AppID	Stores the Application name and version in the following format: `[AppName] [BundleVersion]` For example, `myapp 1.1`
Launch Number	a.Launches	Number of times the application was launched or brought out of the background.
Days since first use	a.DaysSinceFirstUse	Number of days since first run.
Days since last use	a.DaysSinceLastUse	Number of days since last use.
Hour of Day	a.HourOfDay	Measures the hour the app was launched. 24 hour numerical format. Used for time parting to determine peak usage times.
Day of Week	a.DayOfWeek	Number of the week day the app was launched.
Operating System Version	a.OSVersion	OS version.
Days since last upgrade	a.DaysSinceLastUpgrade	Number of days since the application version number has changed. ***Note:*** *Days since last upgrade is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.*
Launches since last upgrade	a.LaunchesSinceUpgrade	Number of launches since the application version number has changed. ***Note:*** *Launches since last upgrade is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.*
Device Name	a.DeviceName	Stores the device name.

Dimension	Analytics Context Data Parameter	Description
		iOS: Comma-separated 2 digit string that Identifies the iOS device. The first number typically represents the device generation, and the second number typically versions different members of the device family. See *iOS Device Versions* for a list of common device names.
Carrier Name	a.CarrierName	Stores the name of the mobile service provider as provided by the device. ***Note:*** *Carrier name is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.*
Resolution	a.Resolution	Width x Height in actual pixels

Using Bloodhound to Test Mobile Applications

Lets you send server calls to a local computer to test mobile applications.

During application development, Bloodhound lets you view server calls locally, and optionally forward the data to Adobe collection servers.

Bloodhound is available for Mac and Windows.

Requirements

• An Intel-based Mac computer running Snow Leopard (10.6) or later, or a Windows PC.
• Network connectivity to your mobile devices or simulators.

Download

See *Bloodhound - App Measurement QA Tool* on Developer Connection.

Using Bloodhound

See the *Bloodhound User Guide*.

Contact and Legal Information

Information to help you contact Adobe and to understand the legal issues concerning your use of this product and documentation.

Help & Technical Support

The Adobe Marketing Cloud Customer Care team is here to assist you and provides a number of mechanisms by which they can be engaged:

- *Check the Marketing Cloud help pages for advice, tips, and FAQs*
- *Ask us a quick question on Twitter @AdobeMktgCare*
- *Log an incident in our customer portal*
- *Contact the Customer Care team directly*
- *Check availability and status of Marketing Cloud Solutions*

Service, Capability & Billing

Dependent on your solution configuration, some options described in this documentation might not be available to you. As each account is unique, please refer to your contract for pricing, due dates, terms, and conditions. If you would like to add to or otherwise change your service level, or if you have questions regarding your current service, please contact your Account Manager.

Feedback

We welcome any suggestions or feedback regarding this solution. Enhancement ideas and suggestions for the Analytics suite *can be added to our Customer Idea Exchange*.

Legal

© 2013 Adobe Systems Incorporated. All Rights Reserved.
Published by Adobe Systems Incorporated.

Terms of Use | *Privacy Center*

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. A trademark symbol (®, ™, etc.) denotes an Adobe trademark.

All third-party trademarks are the property of their respective owners. Updated Information/Additional Third Party Code Information available at *http://www.adobe.com/go/thirdparty*.